Supplementary material for the article: *Offline and online scheduling of concurrent bag-of-tasks applications on heterogeneous platforms*

Anne Benoit, *Member, IEEE,* Loris Marchal, Jean-François Pineau, *Student Member, IEEE,*
Yves Robert, *Fellow, IEEE,* Frédéric Vivien, *Member, IEEE*

CONTENTS

Anne Benoit is with ENS Lyon, University of Lyon and LIP
Loris Marchal is with CNRS, University of Lyon and LIP
Jean-Fran cois Pineau is with ENS Lyon, University of Lyon and LIP
Yves Robert is with ENS Lyon, University of Lyon and LIP
Frédéric Vivien is with INRIA, University of Lyon and LIP
LIP laboratory, UMR 5668, ENS Lyon – CNRS – INRIA – UCBL, Lyon, France

I. Equivalence between non-emptiness of Polyhedron and achievable stretch (Theorem 1)

We first recall the definition of the polyhedron (K) and the meaning of the constraints which makes it:

All tasks sent by the master. The first set of constraints ensures that all the tasks of a given application A_k are actually sent by the master:

$$\forall\ 1 \le k \le n, \sum_{\substack{1\le j \le 2n-1 \\ t_j\ \ge\ r^{(k)} \\ t_{j+1}\ \le\ d^{(k)}}} \sum_{u=1}^{p} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \times (t_{j+1} - t_j) = \Pi^{(k)}. \tag{1}$$

Non-negative buffers. Each buffer should always have a non-negative size:

$$\forall\ 1 \le k \le n, \forall 1 \le u \le p, \forall 1 \le j \le 2n, \quad B_u^{(k)}(t_j) \ge 0. \tag{2}$$

Buffer initialization. At the beginning of the computation of application A_k, all corresponding buffers are empty:

$$\forall\ 1 \le k \le n, \forall 1 \le u \le p, \quad B_u^{(k)}(r^{(k)}) = 0. \tag{3}$$

Emptying Buffer. After the deadline of application A_k, no tasks of this application should remain on any node:

$$\forall\ 1 \le k \le n, \forall 1 \le u \le p, \quad B_u^{(k)}(d^{(k)}) = 0. \tag{4}$$

Task conservation. During time-interval $[t_j, t_{j+1}]$, some tasks of application A_k are received and some are consumed (computed), which impacts the size of the buffer:

$$\forall\ 1 \le k \le n, \forall 1 \le j \le 2n-1, \forall 1 \le u \le p,$$
$$B_u^{(k)}(t_{j+1}) = B_u^{(k)}(t_j) + \left(\rho_{M\to u}^{(k)}(t_j, t_{j+1}) - \rho_u^{(k)}(t_j, t_{j+1})\right) \times \left(t_{j+1} - t_j\right) \tag{5}$$

Bounded computing capacity. The computing capacity of a node should not be exceeded on any time-interval:

$$\forall 1 \le j \le 2n-1, \forall 1 \le u \le p,\ \sum_{k=1}^{n} \rho_u^{(k)}(t_j, t_{j+1}) \frac{w^{(k)}}{s_u^{(k)}}\ \le\ 1. \tag{6}$$

Bounded link capacity. The bandwidth of each link should not be exceeded:

$$\forall 1 \le j \le 2n-1, \forall 1 \le u \le p,\ \sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \le\ 1. \tag{7}$$

Limited sending capacity of master. The total outgoing bandwidth of the master should not be exceeded:

$$\forall 1 \le j \le 2n-1, \quad \sum_{u=1}^{p}\sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{\mathrm{BW}} \le 1. \tag{8}$$

Non-negative throughputs.

$$\forall 1 \le u \le p, \forall 1 \le k \le n, \forall 1 \le j \le 2n-1, \quad \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \ge 0 \text{ and } \ \rho_u^{(k)}(t_j, t_{j+1}) \ge 0. \tag{9}$$

The convex polyhedron (K) can then be defined by the previous constraints.

$$\begin{cases} \rho_{M\to u}^{(k)}(t_j, t_{j+1}), \rho_u^{(k)}(t_j, t_{j+1}),\ \forall k, u, j \text{ such that } 1 \le k \le n, 1 \le u \le p, 1 \le j \le 2n-1 \\ \text{under the constraints (1), (2), (3), (4), (5), (6), (7), (8) and (9)} \end{cases} \tag{K}$$

The problem of the existence of a schedule with maximum stretch $\mathcal{S}$ turns now into checking whether the polyhedron is empty and, if not, into finding a point in the polyhedron, which is expressed by the following Theorem.

Theorem 1: Under the totally fluid model, Polyhedron (K) is not empty if and only if there exists a schedule with stretch $\mathcal{S}$.

Proof: $\boxed{\Rightarrow}$ Assume that the polyhedron is not empty, and consider a point in (K), given by the values of the $\rho_{M\to u}^{(k)}(t_j, t_{j+1})$ and $\rho_u^{(k)}(t_j, t_{j+1})$. We construct a schedule which obeys exactly these values. During time-interval

$[t_j, t_{j+1}]$, the master sends tasks of application A_k to processor P_u with rate $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$, and this processor computes these tasks at a rate $\rho^{(k)}_u(t_j, t_{j+1})$.

To prove that this schedule is valid under the fluid model, and that it has the expected stretch, we define $\rho^{(k)}_{M\to u}(t)$ as the instantaneous communication rate, and $\rho^{(k)}_u(t)$ as the instantaneous computation rate. Then the (fractional) number of tasks of A_k sent to P_u in interval $[0, T]$ is

$$\int_0^T \rho^{(k)}_{M\to u}(t)dt$$

With the same argument as in the previous remark, applied on interval $[0, T]$, we have

$$B^{(k)}_u(T) = \int_0^T \rho^{(k)}_{M\to u}(t)dt - \int_0^T \rho^{(k)}_u(t)dt$$

Since the buffer size is positive for all t_j and evolves linearly in each interval $[t_j, t_{j+1}]$, it is not possible that a buffer has a negative size, so

$$\int_0^T \rho^{(k)}_u(t)dt \le \int_0^T \rho^{(k)}_{M\to u}(t)dt$$

Hence data is always received before being processed.

With the constraints of Polyhedron (K), it is easy to check that no processor or no link is over-utilized and the outgoing capacity of the master is never exceeded. All the deadlines computed for stretch $\mathcal{S}$ are satisfied by construction, so this schedule achieves stretch $\mathcal{S}$.

$\boxed{\Leftarrow}$ Now we prove that if there exists a schedule S_1 with stretch $\mathcal{S}$, Polyhedron (K) is not empty. We consider such a schedule, and we call $\rho^{(k)}_{M\to u}(t)$ (and $\rho^{(k)}_u(t)$) the communication (and computation) rate in this schedule for tasks of application A_k on processor P_u at time t. We compute as follows the average values for communication and computation rates during time interval $[t_j, t_{j+1}]$:

$$\rho^{(k)}_{M\to u}(t_j, t_{j+1}) = \frac{\displaystyle\int_{t_j}^{t_{j+1}} \rho^{(k)}_{M\to u}(t)dt}{t_{j+1} - t_j} \text{ and } \rho^{(k)}_u(t_j, t_{j+1}) = \frac{\displaystyle\int_{t_j}^{t_{j+1}} \rho^{(k)}_u(t)dt}{t_{j+1} - t_j}.$$

In this schedule, all tasks of application A_k are sent by the master, so

$$\int_{r^{(k)}}^{d^{(k)}} \rho^{(k)}_{M\to u}(t)dt = \Pi^{(k)}.$$

With the previous definitions, Equation (1) is satisfied. Along the same line, we can prove that the task conservation constraints (Equation (5)) are satisfied. Constraints on buffers (Equations 3, 4 and 2) are necessarily satisfied by the size of the buffer in schedule S_1 since it is feasible. Similarly, we can check that the constraints on capacities are verified. ■

II. Binary search

The following algorithm describes a binary search to find the optimal stretch.

Algorithm 1: Binary search

```
begin
    S_inf ← 1
    S_sup ← S_max
    while S_sup − S_inf > ϵ do
        S ← (S_sup + S_inf)/2
        if Polyhedron (K) is empty then
            S_inf ← S
        else
            S_sup ← S
    return S_sup
end
```

A. Proof of Theorem 2

The following theorem proves that this algorithm reaches the optimal stretch with a given precision ϵ.

Theorem 2: For any $\epsilon > 0$, Algorithm 1 computes a stretch $\mathcal{S}$ such that there exists a schedule achieving $\mathcal{S}$ and $\mathcal{S} \leq \mathcal{S}_{\text{opt}} + \epsilon$, where $\mathcal{S}_{\text{opt}}$ is the optimal stretch. The complexity of Algorithm 1 is $O(\log \frac{\mathcal{S}_{\max}}{\epsilon})$.

Proof: We prove that at each step, the optimal stretch is contained in the interval $[\mathcal{S}_{\text{inf}}, \mathcal{S}_{\text{sup}}]$ and $\mathcal{S}_{\text{sup}}$ is achievable. This is obvious at the beginning. At each step, we consider the set of constraints for a stretch $\mathcal{S}$ in the interval. If the corresponding polyhedron is empty, Theorem 1 tells us that stretch $\mathcal{S}$ is not achievable, so the optimal stretch is greater than $\mathcal{S}$. If the polyhedron is not empty, there exists a schedule achieving this stretch, thus the optimal stretch is smaller than $\mathcal{S}$.

The size of the work interval is divided by 2 at each step, and we stop when this size is smaller than ϵ. Thus the number of steps is $O(\log \frac{\mathcal{S}_{max}}{\epsilon})$. At the end, $\mathcal{S}_{\text{opt}} \in [\mathcal{S}_{\text{inf}}, \mathcal{S}_{\text{sup}}]$ with $\mathcal{S}_{\text{sup}} - \mathcal{S}_{\text{inf}} \leq \epsilon$, so that $\mathcal{S}_{\text{sup}} \leq \mathcal{S}_{\text{opt}} + \epsilon$, and $\mathcal{S}_{\text{sup}}$ is achievable. ■

B. Binary search with stretch-intervals

In this section, we present another method to compute the optimal stretch in the offline case. This method is based on a linear program built from the constraints of the convex polyhedron (K) with the minimization of the stretch as objective. To do this, we need that other parameters (especially the deadlines) are functions of the stretch. Recall that the deadlines of the applications are computed from their release date and the targeted stretch $\mathcal{S}$:

$$d^{(k)} = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$

Figure 1 represents the evolution of the deadlines $d^{(k)}$ over the targeted stretch $\mathcal{S}$: each deadline is an affine function in $\mathcal{S}$. For the sake of readability, the time is represented on the x axis, and the stretch on the y axis. Special values of stretches $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ are represented on the figure. These *critical values* of the stretch are points where the ordering of the release dates and deadlines of the applications is modified:

- When $\mathcal{S}$ is such a *critical value*, some release dates and deadlines have the same values;
- When $\mathcal{S}$ varies between two such critical values, i.e., when $\mathcal{S}_a < \mathcal{S} < \mathcal{S}_{a+1}$, then the ordering of the release dates and the deadlines is preserved.

To simplify our notations, we add two artificial *critical values* corresponding to the natural bound of the stretch: $\mathcal{S}_1 = 1$ and $\mathcal{S}_m = \infty$.

Our goal is to find the optimal stretch by slicing the stretch space into a number of intervals. Within each interval defined by the *critical values*, the deadlines are linear functions of the stretch. We first show how to find the best stretch within a given interval using a single linear program, and then how to explore the set of intervals with a binary search, so as to find the one containing the optimal stretch.

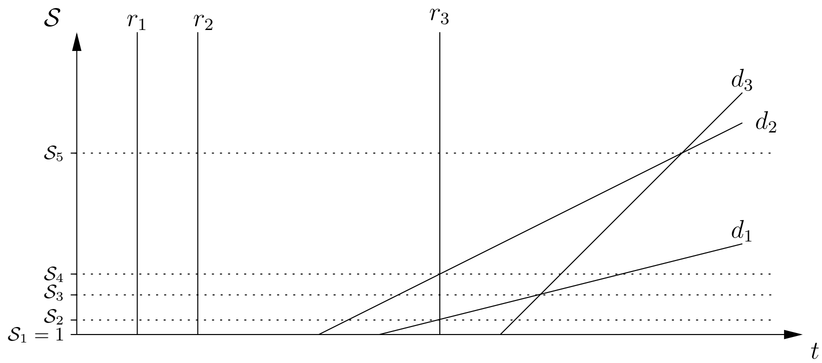


Fig. 1. Relation between stretch and deadlines

1) Within a stretch-interval: In the following, we work on one stretch-interval, called $[\mathcal{S}_a, \mathcal{S}_b]$. For all values of $\mathcal{S}$ in this interval, the release dates $r^{(k)}$ and deadlines $d^{(k)}$ are in a given order, independent of the value of $\mathcal{S}$. As previously, we note $\{t_j\}_{j=1\ldots 2n} = \{r^{(k)}, d^{(k)}\}$, with $t_j \leq t_{j+1}$. As the values of the t_j may change when $\mathcal{S}$ varies, we write $t_j = \alpha_j \mathcal{S} + \beta_j$. This notation is general enough for all $r^{(k)}$ and $d^{(k)}$:

- If $t_j = r^{(k)}$, then $\alpha_j = 0$ and $\beta_j = r^{(k)}$.
- If $t_j = d^{(k)}$, then $\alpha_j = MS^{*(k)}$ and $\beta_j = r^{(k)}$.

Note that like previously, some t_j might be equal, and especially when the stretch reaches a bound of the stretch-interval ($\mathcal{S} = \mathcal{S}_a$ or $\mathcal{S} = \mathcal{S}_b$), that is a critical value. For the sake of simplicity, we do not try to discard the empty time-intervals, to avoid the renumbering of the epochal times.

When we rewrite the constraints defining the convex polyhedron (K) with these new notations, we obtain quadratic constraints instead of linear constraints. To avoid this, we introduce new notations. Instead of considering the instantaneous communication and computation rates, we use the total amount of tasks sent or computed during a given time-interval. Formally we define $A^{(k)}_{M\to u}(t_j, t_{j+1})$ to be the fractional number of tasks of application A_k sent by the master to processor P_u during the time-interval $[t_j, t_{j+1}]$. Similarly, we denote by $A^{(k)}_u(t_j, t_{j+1})$ the fractional number of tasks of application A_k computed by processor P_u during the time-interval $[t_j, t_{j+1}]$. Of course, these quantities are linked to our previous variables. Indeed, we have:

$$\begin{aligned} A^{(k)}_{M\to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\ A^{(k)}_u(t_j, t_{j+1}) &= \rho^{(k)}_u(t_j, t_{j+1}) \times (t_{j+1} - t_j) \end{aligned}$$

with $t_{j+1} - t_j = (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)$.

We also introduce $\mathcal{I}^{(\|)}$, the set of time-intervals where application A_k can be executed:

$$\mathcal{I}^{(\|)} = \{[t_j, t_{j+1}], \text{ such that } t_j \geq r^{(k)} \text{ and } t_{j+1} \leq d^{(k)}\}$$

Note that for the stretch range $[\mathcal{S}_a, \mathcal{S}_b]$ where we are working, these sets of time-intervals does not change even if the bounds of the time-intervals vary.

We rewrite the set of constraints with these new notations:

Total number of tasks. We make sure that all tasks of application A_k are sent by the master:

$$\forall\ 1 \leq k \leq n, \qquad \sum_{\substack{1\leq j\leq 2n-1 \\ t_j \ \geq\ r^{(k)} \\ t_{j+1}\ \leq\ d^{(k)}}} \sum_{u=1}^{p} A^{(k)}_{M\to u}(t_j, t_{j+1}) = \Pi^{(k)} \tag{10}$$

Non-negative buffer. Each buffer should always have a non-negative size:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \forall 1 \leq j \leq 2n, \quad B^{(k)}_u(t_j) \geq 0 \tag{11}$$

Buffer initialization. At the beginning of the computation of application A_k, all corresponding buffers are empty:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \quad \text{for } t_j = r^{(k)}, \quad B_u^{(k)}(t_j) = 0 \tag{12}$$

Emptying Buffer. After the deadline of application A_k, no tasks of this application should remain on any node:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \quad \text{for } t_j = d^{(k)}, \quad B_u^{(k)}(t_j) = 0 \tag{13}$$

Task conservation. During time-interval $[t_j, t_{j+1}]$, some tasks of application A_k are received and some are consumed (computed), which impacts the size of the buffer:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p, \quad B_u^{(k)}(t_{j+1}) = B_u^{(k)}(t_j) + A_{M\to u}^{(k)}(t_j, t_{j+1}) - A_u^{(k)}(t_j, t_{j+1}) \tag{14}$$

Bounded computing capacity. The computing capacity of a node should not be exceeded on any time-interval:

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p, \sum_{k=1}^{n} A_u^{(k)}(t_j, t_{j+1}) \frac{w^{(k)}}{s_u^{(k)}} \ \leq\ (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j) \tag{15}$$

Bounded link capacity. The bandwidth of each link should not be exceeded:

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p, \sum_{k=1}^{n} A_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \leq\ (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j) \tag{16}$$

Limited sending capacity of master. The total outgoing bandwidth of the master should not be exceeded:

$$\forall 1 \leq j \leq 2n-1, \sum_{u=1}^{p} \sum_{k=1}^{n} A_{M\to u}^{(k)}(t_j, t_{j+1}) \delta^{(k)} \leq \text{BW} \times \big((\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)\big) \tag{17}$$

We also add a constraint to bound the objective stretch to be in the targeted stretch-interval:

$$\mathcal{S}_a \leq \mathcal{S} \leq \mathcal{S}_b \tag{18}$$

Even if the bounds of the sum on the time-intervals in Equation (10) seem to depend on $\mathcal{S}$, the set of intervals involved in the sum does not vary as the order of the t_j values is fixed for $\mathcal{S}_a \leq \mathcal{S} \leq \mathcal{S}_b$. With the objective of minimizing the stretch, we get the following linear program.

$$\text{(LP)} \begin{cases} \text{MINIMIZE } \mathcal{S}, \\ \text{UNDER THE CONSTRAINTS } (10),\ (11),\ (12),\ (13),\ (14),\ (15),\ (16),\ (17),\ (18) \end{cases}$$

Solving this linear program allows to find the minimum possible stretch in the stretch-interval $[\mathcal{S}_a, \mathcal{S}_b]$. If the minimum stretch computed by the linear program is $\mathcal{S}_{\text{opt}} > \mathcal{S}_a$, this means that there is not better possible stretch in $[\mathcal{S}_a, \mathcal{S}_b]$, and thus there is no better stretch for all possible values. On the contrary, if $\mathcal{S}_{\text{opt}} = \mathcal{S}_a$, we cannot conclude: $\mathcal{S}_a$ may be the optimal stretch, or the optimal stretch is smaller than $\mathcal{S}_a$. In this case, the binary search is continued with smaller stretch values. At last, if there is no solution to the linear program, then there exists no possible stretch smaller or equal to $\mathcal{S}_b$, and the binary search is continued with larger stretch values. This binary search and its proof are described below.

When $\mathcal{S}_a < \mathcal{S}_{\text{opt}} \leq \mathcal{S}_b$, we can prove that $\mathcal{S}_{\text{opt}}$ is the optimal stretch.

Theorem 3: The linear program (LP) finds the optimal stretch provided that the optimal stretch is in $]\mathcal{S}_a, \mathcal{S}_b]$.

Proof: The proof highly depends on Theorem 1. First, consider an optimal solution of the linear program (LP). We compute

$$\rho_{M\to u}^{(k)}(t_j, t_{j+1}) = \frac{A_{M\to u}^{(k)}(t_j, t_{j+1})}{(\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)} \quad \text{and} \quad \rho_u^{(k)}(t_j, t_{j+1}) = \frac{A_u^{(k)}(t_j, t_{j+1})}{(\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)}.$$

These variables constitute a valid solution of the set of constraints of Theorem 1 for $\mathcal{S} = \mathcal{S}_{\text{opt}}$. Therefore there exists a schedule achieving stretch $\mathcal{S}_{\text{opt}}$.

Assume now that there exists a schedule with stretch $\mathcal{S}$ such that $\mathcal{S}_a < \mathcal{S} < \mathcal{S}_b$. Due to Theorem 1, there exists values for $\rho_{M\to u}^{(k)}(t_j, t_{j+1})$ and $\rho_u^{(k)}(t_j, t_{j+1})$ satisfying the corresponding set of constraints for $\mathcal{S}$. Then we compute

$$\begin{aligned} A^{(k)}_{M\to u}(t_j,t_{j+1}) &= \rho^{(k)}_{M\to u}(t_j,t_{j+1}) \times \big((\alpha_{j+1}-\alpha_j)\mathcal{S} + (\beta_{j+1}-\beta_j)\big) \\ A^{(k)}_{u}(t_j,t_{j+1}) &= \rho^{(k)}_{u}(t_j,t_{j+1}) \times \big((\alpha_{j+1}-\alpha_j)\mathcal{S} + (\beta_{j+1}-\beta_j)\big) \end{aligned}$$

$A^{(k)}_{M\to u}(t_j,t_{j+1})$ and $A^{(k)}_{u}(t_j,t_{j+1})$ constitute a solution of the linear program (LP) with objective value $\mathcal{S}$. As the objective value $\mathcal{S}_{\text{opt}}$ found by the linear program is minimal among all possible solutions, we have $\mathcal{S}_{\text{opt}} \le \mathcal{S}$. ■

2) Binary search among stretch intervals: We assume that we have computed the bounds of the stretch intervals: $\mathcal{S}_1,\dots,\mathcal{S}_m$. The binary search to reach the optimal stretch works as follows:

Algorithm 2: Binary search among stretch-intervals

```
begin
    L ← 1 and U ← max
    while U − L > 1 do
        M ← ⌊(L + U)/2⌋
        Solve the linear program (LP) for interval [S_M, S_{M+1}]
        if there is a solution with objective value S_opt then
            if S_opt > S_M then
                return S_opt
            else
                U ← M
        else
            L ← M
    Solve the linear program (LP) for interval [S_L, S_U]
    return the objective value S_opt of the solution
end
```

Theorem 4: Algorithm 2 finds the optimal stretch value in a polynomial number of steps.

Proof: This algorithm performs a binary search among the m stretch-intervals. Thus, the number of steps of this search is $O(\log m)$ and each step consists in solving a linear program, which can be done in polynomial time.

We prove that the optimal stretch is always contained in the interval $[\mathcal{S}_L,\mathcal{S}_U]$. This is obviously true in the beginning. On a stretch-interval $[\mathcal{S}_M,\mathcal{S}_{M+1}]$, the minimum possible stretch $\mathcal{S}_{\text{opt}}$ is computed. If $\mathcal{S}_{\text{opt}} > \mathcal{S}_M$, thanks to Theorem 3, we know that $\mathcal{S}_{\text{opt}}$ is the optimal stretch. If there is no solution, no stretch values in the stretch-interval $[\mathcal{S}_M,\mathcal{S}_{M+1}]$ is feasible, so the optimal stretch is in $[\mathcal{S}_{M+1},\mathcal{S}_U]$. If $\mathcal{S}_{\text{opt}} = \mathcal{S}_M$, then the optimal stretch smaller or equal than $\mathcal{S}_M$. Thus, the optimal stretch is still contained in $[\mathcal{S}_M,\mathcal{S}_{M+1}]$ after one iteration. If we exit *while* loop without having return the optimal stretch, then $U = L+1$ and the optimal stretch is contained in the stretch-interval $[\mathcal{S}_L,\mathcal{S}_U]$. We compute this value with the linear program and return it. ■

III. QUASI-OPTIMALITY FOR MORE REALISTIC BOUNDED MULTIPORT MODELS

In this section, we explain how the previous optimality result can be adapted to the other bounded multiport models presented in Section II-A.3 of the manuscript. As expected, the more realistic the model, the less tight the optimality guaranty. Fortunately, we are always able to reach *asymptotic optimality*: our schedules get closer to the optimal as the number of tasks per application increases.

We describe the delay induced by each model in comparison to the fluid model: starting from a schedule optimal under the fluid model (BMP-FC-SS), the idea to build a schedule with comparable performance under a more constrained scenario.

In the following, we consider a schedule S_1, with stretch $\mathcal{S}$, valid under the totally fluid model (BMP-FC-SS). For the sake of simplicity, we consider that this schedule has been built from a point in Polyhedron (K) as explained in the previous section: the computation and communication rates ($\rho_u^{(k)}(t_j, t_{j+1})$ and $\rho_{M\to u}^{(k)}(t_j, t_{j+1})$) are constant during each interval, and are defined by the coordinates of the point in Polyhedron (K).

We assess the *delay* induced by each model. Given the stretch $\mathcal{S}$, we can compute a deadline $d^{(k)}$ for each application A_k. By moving to more constrained models, we will not be able to ensure that the finishing time $MS^{(k)}$ is smaller than $d^{(k)}$. We call lateness for application A_k the quantity $\max\{0, MS^{(k)} - d^{(k)}\}$, that is the time between the due date of an application and its real termination. Once we have computed the maximum lateness for each model, we show how to obtain asymptotic optimality in Section III-C.

A. Without simultaneous start: the BMP-FC model

We consider here the BMP-FC model, which differs from the previous model only by the fact that a task cannot start before it has been totally received by a processor.

Theorem 5: From schedule S_1, we can build a schedule S_2 obeying the BMP-FC model where the maximum lateness for each application is $\displaystyle\max_{1\le u\le p}\sum_{k=1}^{n}\frac{w^{(k)}}{s_u^{(k)}}$.

Proof: From the schedule S_1, valid under the fluid model (BMP-FC-SS), we aim at building S_2 with a similar stretch where the execution of a task cannot start before the end of the corresponding communication. We first build a schedule as follows, for each processor P_u ($1 \le u \le p$):

1) Communications to P_u are the same as in S_1;
2) By comparison to S_1, the computations on P_u are shifted for each application A_k: the computation of the first task of A_k is not really performed (P_u is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.

Because of the shift of the computations, the last task of application A_k is not executed in this schedule at time $d^{(k)}$. We complete the construction of S_2 by adding some delay after deadline $d^{(k)}$ to process this last task of application A_k at full speed, which takes a time $\frac{w^{(k)}}{s_u^{(k)}}$. All the following computations on processor P_u (in the next time-intervals) are shifted by this delay.

The lateness for any application A_k on processor P_u is at most the sum of the delays for all applications on this processor, $\sum_{k=1}^{n}\frac{w^{(k)}}{s_u^{(k)}}$, and the total lateness of A_k is bounded by the maximum lateness between all processors:

$$\textit{lateness}^{(k)} \le \max_{1\le u\le p}\sum_{k=1}^{n}\frac{w^{(k)}}{s_u^{(k)}}$$

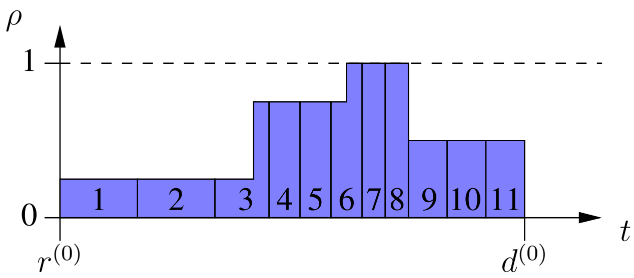
An example of such a schedule S_2 is shown on Figure 2 (on a single processor). ■

B. Atomic execution of tasks: the BMP-AC model

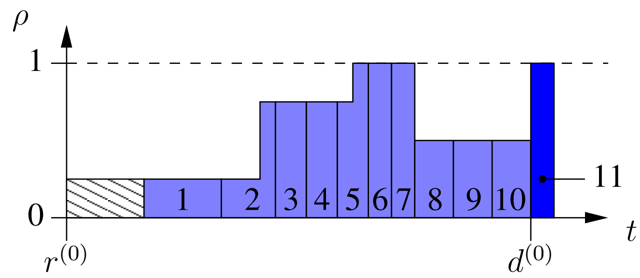
We now move to the BMP-AC model, where a given processor cannot compute several tasks in parallel, and the execution of a task cannot be preempted: a started task must be completed before any other task can be processed.

Theorem 6: From schedule S_1, we can build a schedule S_3 obeying the BMP-AC model where the maximum lateness for each application is

$$\max_{1\le u\le p} 2n \times \sum_{k=1}^{n}\frac{w^{(k)}}{s_u^{(k)}}.$$



(a) Schedule S_1 (BMP-FC-SS model)



(b) Schedule S_2 (BMP-FC model)

Fig. 2. Example of the construction of a schedule S_2 for BMP-FC model from a schedule S_1 for BMP-FC-SS model. We plot only the computing rate. Each box corresponds to the execution of one task.

Proof: Starting from a schedule S_1 valid under the fluid model (BMP-FC-SS), we want to build S_3, valid in BMP-AC. We take here advantage of the properties described in Section IV-A of one-dimensional load-balancing schedules, and especially of S_{1D}^{-2}. Schedule S_3 is built as follows:

1) Communications are kept unchanged;
2) We consider the computations taking place in S_1 on processor P_u during time-interval $[t_j, t_{j+1}]$. A rational number of tasks of each application may be involved in the fluid schedule. We first compute the integer number of tasks of application A_k to be computed in S_3:
$$n_{u,j,k} = \left\lfloor \rho_u^{(k)}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \right\rfloor.$$
The first $n_{u,j,k}$ tasks of A_k scheduled in time-interval $[t_j, t_{j+1}]$ on P_u are organized using the transformation to build S_{1D}^{-2} in Section IV-A.
3) Then, the computations are shifted as for S_2: for each application A_k, the computation of the first task of A_k is not really performed (the processor is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.

Lemma 2 proves that, during time-interval $[t_j, t_{j+1}]$, on processor P_u, a computation does not start earlier in S_3 than in S_1. As S_1 obeys the totally fluid model (BMP-FC-SS), a computation of S_1 does not start earlier than the corresponding communication, so a computation of task i of application A_k in S_1 does not start earlier than the finish time of the communication for task $i-1$ of A_k. Together with the shifting of the computations, this proves that in S_3, the computation of a task does not start earlier than the end of the corresponding communication, on each processor.

Because of the rounding down to the closest integer, on each processor P_u, at each time-interval, S_3 computes at most one task less than S_1 of application A_k. Moreover, one more task computation of application A_k is not performed in S_3 due to the computation shift. On the whole, as there are at most $2n-1$ time-intervals, at most $2n$ tasks of A_k remain to be computed on P_u at time $d^{(k)}$. The delay for application A_k is:

$$\textit{lateness}^{(k)} \leq \max_{1 \leq u \leq p} 2n \times \sum_{k=1}^{n} \times \frac{w^{(k)}}{s_u^{(k)}}.$$

■

This is obviously not the most efficient way to construct a schedule for the BMP-AC model: in particular, each processor is idle during each interval (because of the rounding down). It would certainly be more efficient to sometimes start a task even if it cannot be terminated before the end of the interval. This is why for our experiments, we implemented on each worker a greedy schedule with Earliest Deadline First Policy instead of this complex construction. However, we can easily prove that this construction has an asymptotic optimal stretch, unlike other greedy strategies.

C. Asymptotic optimality

For the sake of the completeness of this section, we recall the motivation and the definition of the asymptotic optimality, which are described in the manuscript, before detailing the proof of Theorem 7.

In this section, we show that the previous schedules are close to the optimal, when applications are composed of a large number of tasks. To establish such an asymptotic optimality, we have to prove that the gap computed above gets smaller when the number of tasks gets larger. At first sight, we would have to study the limit of the application stretch when $\Pi^{(k)}$ is large for each application. However, if we simply increase the number of tasks in each application without changing the release dates and the tasks characteristics, then the problem will look totally different: any schedule will run for a very long time, and the time separating the release dates will be negligible in front of the whole duration of the schedule. This behavior is not meaningful for our study.

To study the asymptotic behavior of the system, we rather change the granularity of the tasks: we show that when applications are composed of a large number of small-size tasks, then the maximal stretch is close to the optimal one obtained with the fluid model. To take into account the application characteristics, we introduce the granularity g, and we redefine the application characteristics with this new variable:

$$\Pi_g^{(k)} = \frac{\Pi^{(k)}}{g}, \qquad w_g^{(k)} = g \times w^{(k)} \quad \text{and} \quad \delta_g^{(k)} = g \times \delta^{(k)}.$$

When $g = 1$, we get back to the previous case. When $g < 1$, there are more tasks but they have smaller communication and computation size. For any g, the total communication and computation amount per application is kept the same, thus it is meaningful to consider the original release dates.

Our goal is to study the case $g \to 0$. Note that under the totally fluid model (BMP-FC-SS), the granularity has no impact on the performance (or the stretch). Indeed, the fluid model can be seen as the extreme case where $g = 0$. The optimal stretch under the BMP-FC-SS $\mathcal{S}_{\text{opt}}$ does not depend on g.

Theorem 7: When the granularity is small, the schedule constructed above for the BMP-FC (respectively BMP-AC) model is asymptotically optimal for the maximum stretch, that is

$$\lim_{g \to 0} \mathcal{S} = \mathcal{S}_{\text{opt}}$$

where $\mathcal{S}$ is the stretch of the BMP-FC (resp. BMP-AC) schedule, and $\mathcal{S}_{\text{opt}}$ the stretch of the optimal fluid schedule.

Proof: The lateness of the applications computed in Section III-A for the BMP-FC model, and in Section III-B for the BMP-AC model, becomes smaller when the granularity increase: for the BMP-FC model, we have

$$\textit{lateness}^{(k)} \leq \max_{1 \leq u \leq p} \sum_{k=1}^{n} \frac{w_g^{(k)}}{s_u^{(k)}} \xrightarrow[g \to 0]{} 0.$$

Similarly, for the BMP-AC model,

$$\textit{lateness}^{(k)} \leq \max_{1 \leq u \leq p} 2n \times \sum_{k=1}^{n} \frac{w_g^{(k)}}{s_u^{(k)}} \xrightarrow[g \to 0]{} 0.$$

Thus, when g gets close to 0, the stretch obtained by these schedules is close to $\mathcal{S}_{\text{opt}}$. ■

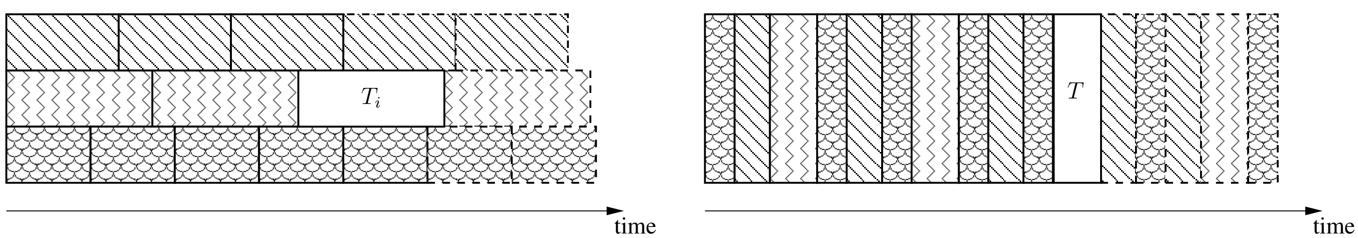
IV. Asymptotic optimality for the one-port model

A. Property of the one-dimensional load-balancing schedule

In this section, we introduce the one-dimension load-balancing algorithm, and interesting properties that can be derived from schedules obtained using this algorithm.

In the next section, we compare the results obtained under the different communication and computation models introduced in the manuscript. One of the major differences between these models is whether they allow –or not– preemption and time-sharing. On the one hand, we study "fluid" models, where a resource (processor or communication link) can be simultaneously used by several tasks, provided that the total utilization rate is below one. On the other hand, we also study "atomic" models, where a resource can be devoted to only one task, which cannot be preempted: once a task is started on a given resource, this resource cannot perform other tasks before the first one is completed. In this section, we show how to construct a schedule without preemption from fluid schedules, in a way that keeps the interesting properties of the original schedule. Namely, we aim at constructing atomic-model schedules in which tasks terminate not later, or start not earlier, than in the original fluid schedule.

We consider a general case of n applications $A_1, \dots, A_n$ to be scheduled on the same resource, typically a given processor, and we denote by t_k the time needed to process one task of application A_k at full speed. We start from a fluid schedule S_{fluid} where each application A_k is devoted a share α_k of the resource, such that $\sum_{k=1}^{n} \alpha_k \leq 1$. Figure 3(a) illustrates such a schedule.



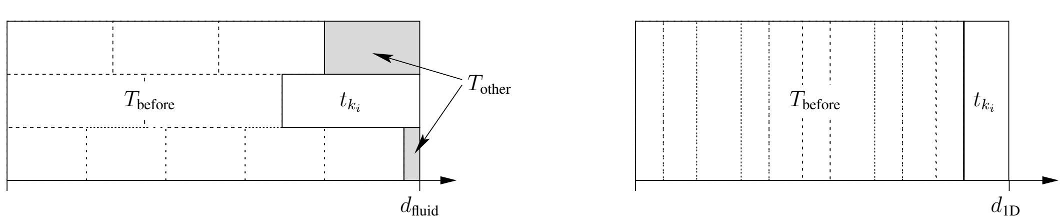
(a) fluid schedule S_{fluid} (b) atomic schedule S_{1D}

Fig. 3. Gantt charts for the proof illustrating the one-dimensional load-balancing algorithm.

From S_{fluid}, we build an atomic-model schedule S_{1D} using a one-dimensional load-balancing algorithm [1], [2]: at any time step, if n_k is the number of tasks of application A_k that have already been scheduled, the next task to be scheduled is the one which minimizes the quantity $\frac{(n_k+1)\times t_k}{\alpha_k}$. Figure 3(b) illustrates the schedule obtained. We now prove that this schedule has the nice property that a task is not processed later in S_{1D} than in S_{fluid}.

Lemma 1: In the schedule S_{1D}, a task T does not terminate later than in S_{fluid}.

Proof: First, we point out that t_k/α_k is the time needed to process one task of application A_k in S_{fluid} (with rate α_k). So $\frac{n_k \times t_k}{\alpha_k}$ is the time needed to process the first n_k tasks of application A_k. The scheduling decision which chooses the application minimizing $\frac{(n_k+1)\times t_k}{\alpha_k}$ consists in choosing the task which is not yet scheduled and which terminates first in S_{fluid}. Thus, in S_{1D}, the tasks are executed in the order of their termination date in S_{fluid}. Note that if several tasks terminate at the very same time in S_{fluid}, then these tasks can be executed in any order in S_{1D}, and the partial order of their termination date is still observed in S_{1D}.



Then, consider a task T_i of a given application A_{k_i}, its termination date d_{fluid} in S_{fluid}, and its termination date d_{1D} in S_{1D}. We call $\mathcal{S}_{\text{before}}$ the set of tasks which are executed before T_i in S_{1D}. Because S_{1D} executes the tasks in the order of their termination date in S_{fluid}, $\mathcal{S}_{\text{before}}$ is made of tasks which are completed before T_i in S_{fluid}, and possibly some tasks completed at the same time as T_i (at time d_{fluid}). We denote by T_{before} the time needed to process the tasks in $\mathcal{S}_{\text{before}}$.

In S_{1D}, we have $d_{\text{1D}} = T_{\text{before}} + t_{k_i}$ whereas in S_{fluid}, we have $d_{\text{fluid}} = T_{\text{before}} + t_{k_i} + T_{\text{other}}$ where T_{other} is the time spent processing tasks from other application than A_k and which are not completed at time d_{fluid}, or tasks completing at time d_{fluid} and scheduled later than T_i in S_{1D}. In S_{1D}, we have $d_{\text{1D}} = T_{\text{before}} + t_{k_i}$. Since $T_{\text{other}} \geq 0$, we have $d_{\text{1D}} \leq d_{\text{fluid}}$. ■

The previous property is useful when we want to construct an atomic-model schedule, that is a schedule without preemption, in which task results are available no later than in a fluid schedule. On the contrary, it can be useful to ensure that no task will start earlier in an atomic-model schedule than in the original fluid schedule. Here is a procedure to construct a schedule with the latter property.

1) We start again from a fluid schedule S_{fluid}, of makespan M. We transform this schedule into a schedule S_{fluid}^{-1} by reversing the time: a task beginning at time b and finishing at time f in S_{fluid} is scheduled to start at time $M - f$ and to terminate at $M - b$ in S_{fluid}^{-1}, and is processed at the same rate as in S_{fluid}. Note that this is possible since we have no precedence constraints between tasks.
2) Then, we apply the previous one-dimensional load-balancing algorithm on S_{fluid}^{-1}, leading to the schedule S_{1D}^{-1}. Thanks to the previous result, we know that a task T does not terminate later in S_{1D}^{-1} than in S_{fluid}^{-1}.
3) Finally, we transform S_{1D}^{-1} by reverting the time one last time: we obtain the schedule S_{1D}^{-2}. A task beginning at time b and finishing at time f in S_{1D}^{-1} starts at time $M - f$ and finishes at time $M - b$ in S_{1D}^{-2}. Note that S_{1D}^{-1} may have a makespan smaller that M (if the resource was not totally used in the original schedule S_{fluid}). In this case, our method automatically introduces idle time in the one-dimensional schedule, to avoid that a task is started too early.

Lemma 2: A task does not start sooner in S_{1D}^{-2} than in S_{fluid}.

Proof: Consider a task T, call f_1 its termination date in S_{fluid}^{-1}, and f_2 its termination date in S_{1D}^{-1}. Thanks to Lemma 1, we know that $f_2 \leq f_1$. By construction of the reverted schedules, the starting date of task T in S_{fluid} is $M - f_1$. Similarly, its starting date in S_{1D}^{-2} is $M - f_2$ and we have $M - f_2 \geq M - f_1$. ■

B. Asymptotic optimality

In this section, we explain how to modify the study on multiport models to cope with the one-port model. We cannot simply extend the result obtained for the fluid model to the one-port model (as we have done for the multiport models) since the parameters for modeling communications are not the same. Actually, the one-port model limits the time spent by a processor (here the master) to send data whereas the multiport model limits its bandwidth capacity. Thus, we have to modify the corresponding constraints. Constraint (8) is replaced by the following one.

$$\forall 1 \leq j \leq 2n-1, \sum_{u=1}^{p} \sum_{k=1}^{n} \rho_{M \to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \leq 1 \tag{8-b}$$

Note that the only difference with Constraint (8) is that, now, we bound the time needed by the master to send all data instead of the volume of the data itself. The set of constraints corresponding to the scheduling problem under the one-port model, for a maximum stretch $\mathcal{S}$, are gathered by the definition of Polyhedron (K_1):

$$\begin{cases} \rho_{M \to u}^{(k)}(t_j, t_{j+1}), \rho_u^{(k)}(t_j, t_{j+1}), \ \forall k, u, j \text{ such that } 1 \leq k \leq n, 1 \leq u \leq p, 1 \leq j \leq 2n-1 \\ \text{under the constraints } (1), (5), (3), (4), (2), (6), (7), \text{(8-b), and } (9) \end{cases} \tag{K_1}$$

As previously, the existence of a point in the polyhedron is linked to the existence of a schedule with stretch $\mathcal{S}$. However, we have no fluid model which could perfectly follow the behavior of the linear constraints. Thus we only target asymptotic optimality.

Theorem 8: (a) If there exists a schedule valid under the one-port model with stretch $\mathcal{S}_1$, then Polyhedron (K_1) is not empty for $\mathcal{S}_1$.

(b) Conversely, if Polyhedron (K_1) is not empty for the stretch objective $\mathcal{S}_2$, then there exists a schedule valid for the problem under the one-port model with parameters $\Pi_g^{(k)}$, $\delta_g^{(k)}$, and $w_g^{(k)}$, as defined in Section III-C, whose stretch $\mathcal{S}$ is such that

$$\lim_{g \to 0} \mathcal{S} = \mathcal{S}_2.$$

Proof:

(a) To prove the first part of the theorem, we prove that for any schedule with stretch $\mathcal{S}_1$, we can construct a point in Polyhedron (K_1). Given such a schedule, we denote by $A^{(k)}_{M\to u}(t_j, t_{j+1})$ the total number of tasks of application A_k sent by the master to processor P_u during interval $[t_j, t_{j+1}]$. Note that this may be a rational number if there are ongoing transfers at times t_j and/or t_{j+1}. Similarly, we denote by $A^{(k)}_u(t_j, t_{j+1})$ the total (rational) number of tasks of A_k processed by P_u during interval $[t_j, t_{j+1}]$. Then we compute:

$$\rho^{(k)}_{M\to u}(t_j, t_{j+1}) = \frac{A^{(k)}_{M\to u}(t_j, t_{j+1})}{t_{j+1} - t_j} \quad \text{and} \quad \rho^{(k)}_u(t_j, t_{j+1}) = \frac{A^{(k)}_u(t_j, t_{j+1})}{t_{j+1} - t_j}.$$

As in the fluid case, we can also compute the state of the buffers based on these quantities:

$$B^{(k)}_u(t_j) = \sum_{t_i+1 \le t_j} A^{(k)}_{M\to u}(t_i, t_{i+1}) - A^{(k)}_u(t_i, t_{i+1})$$

We can easily check that all constraints (1),(2), (3), (4), (5), (6), (7), and (8-b) are satisfied. Variables $B^{(k)}_u(t_j)$, $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$, and $\rho^{(k)}_u(t_j, t_{j+1})$ define a point in Polyhedron (K_1).

(b) From a point in Polyhedron (K_1), we build a schedule which is asymptotically optimal, as defined in Section III-C. During each interval $[t_j, t_{j+1}]$, for each worker P_u, we proceed as follows.

1) We first consider a fluid-model schedule S_f following exactly the rates defined by the point in the polyhedron: the tasks of application A_k are sent with rate $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$ and processed at rate $\rho^{(k)}_u(t_j, t_{j+1})$.

2) We transform both the communication schedule and the computation schedule using one-dimensional load-balancing algorithms. We first compute the integer number of tasks that can be sent in the one-port schedule:

$$n^{\text{comm}}_{u,j,k} = \left\lfloor \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \right\rfloor.$$

The number of tasks that can be computed on P_u in this time-interval is bounded both by the number of tasks processed in the fluid-model schedule, and by the number of tasks received during this time-interval plus the number of remaining tasks:

$$n^{\text{comp}}_{u,j,k} = \min \left\{ \left\lfloor \rho^{(k)}_u(t_j, t_{j+1}) \times (t_{j+1} - t_j) \right\rfloor \;,\; n^{\text{comm}}_{u,j,k} + \sum_{i=1}^{j-1} \left(n^{\text{comm}}_{u,i,k} - n^{\text{comp}}_{u,i,k} \right) \right\}$$

Consider a fluid-model schedule based on the value $\rho^{(k)}_u(t_j, t_{j+1})$ and $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$, for each time-interval $[t_i, t_{i+1}]$ preceding the current one ($i \le j$). By rounding down the number of tasks received by P_u in each of these time-intervals, we potentially decrease the available number of tasks available for computation by j (the maximum number of preceding time-intervals). This allows us to give an upper bound for the difference between the theoretical number of tasks processed (in the fluid-model schedule) and the actual number, under the one-port model:

$$n^{\text{comp}}_{u,j,k} \ge \rho^{(k)}_u(t_j, t_{j+1}) \times (t_{j+1} - t_j) - j$$

The first $n^{\text{comm}}_{u,j,k}$ tasks sent in schedule S_f are organized with the one-dimensional load-balancing algorithm into S_{1D}, while the last $n^{\text{comp}}_{u,j,k}$ tasks executed in schedule S_f are organized with the inverse one-dimensional load-balancing algorithm S^{-2}_{1D} (see Section IV-A).

3) Then, the computations are shifted: for each application A_k, the computation of the first task of A_k is not really performed (the processor is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.

The proof of the validity of the obtained schedule is very similar to the proof of Theorem 6 for the BMP-AC model: we use the fact that a task does not start earlier in S^{-2}_{1D} than in S_f, and no later in S_{1D} than in S_f to prove that the data needed for the execution of a given task are received in time.

At time $d^{(k)}$, some tasks of application A_k are still not processed, and some may even not be received yet. Let us denote by L_k the number of time-intervals between $r^{(k)}$ and $d^{(k)}$, that is time-intervals where tasks of application A_k may be processed ($L_k \le 2n - 1$). Because of the rounding of the numbers of tasks sent, at most one task is not transmitted in each interval, for each application. At time $d^{(k)}$, we thus have at most L_k

tasks of application A_k to be sent to each processor P_u. We have to serialize the sending operations, which takes a time at most

$$\sum_{u=1}^{n} \frac{L_k \times \delta^{(k)}}{b_u}$$

Then, the number of tasks remaining to be processed on processor P_u is upper bounded by $2L_k+1$: at most L_k are received late because of the rounding of the number of tasks received, at most L_k tasks are received but not computed because we also round the number of tasks processed, and one more task may also remain because of the computation shift. The computation (at full speed) of all these tasks takes at most a time $(2L_k+1)\frac{w^{(k)}}{s_u^{(k)}}$ on processor P_u. Overall, the delay induced on all processors for finishing application A_k can be bounded by:

$$\sum_{u=1}^{n} \frac{L_k \times \delta^{(k)}}{b_u} + \max_{1 \leq u \leq p} (2L_k + 1) \times \frac{w^{(k)}}{s_u^{(k)}}.$$

As $L_k \leq 2n-1$, the lateness of any application A_k is thus:

$$\textit{lateness}^{(k)} \leq \sum_k \left(\sum_{u=1}^{n} \frac{(2n-1) \times \delta^{(k)}}{b_u} + \max_{1 \leq u \leq p} (4n-1) \times \frac{w^{(k)}}{s_u^{(k)}} \right).$$

As in the proof of Theorem 7, when the granularity becomes small, the stretch of the obtained schedule becomes as close to $\mathcal{S}_2$ as we want.

■

V. Complete simulation results

In Figures 4, 5, 6 and 7, we present the complete simulation results regarding metrics that are not our main objective, that is: sum-stretch, makespan, max-flow and sum-flow.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	2.064	6.783	(± 3.210)	30.7	(the best in 0.0 %)
FIFO_MCT	1.322	2.754	(± 0.670)	6.45	(the best in 0.0 %)
FIFO_DD	2.064	6.783	(± 3.210)	30.7	(the best in 0.0 %)
SPT_RR	1.019	2.942	(± 1.221)	10.1	(the best in 0.0 %)
SPT_MCT	1.000	1.182	(± 0.183)	2.53	(the best in 2.4 %)
SPT_DD	1.019	2.942	(± 1.221)	10.1	(the best in 0.0 %)
SRPT_RR	1.007	2.607	(± 1.071)	8.93	(the best in 0.0 %)
SRPT_MCT	**1.000**	**1.045**	**(± 0.098)**	**1.92**	**(the best in 25.5 %)**
SRPT_DD	1.007	2.607	(± 1.071)	8.93	(the best in 0.0 %)
SWRPT_RR	1.000	2.596	(± 1.068)	8.96	(the best in 0.1 %)
SWRPT_MCT	**1.000**	**1.038**	**(± 0.098)**	**1.92**	**(the best in 60.1 %)**
SWRPT_DD	1.000	2.596	(± 1.068)	8.96	(the best in 0.1 %)
MWMA_NBT	1.051	2.013	(± 0.644)	5.41	(the best in 0.0 %)
MWMA_MS	1.663	4.183	(± 1.269)	11.5	(the best in 0.0 %)
CBS3M_FIFO_ONLINE	1.000	1.294	(± 0.208)	2.16	(the best in 0.4 %)
CBS3M_EDF_ONLINE	**1.000**	**1.201**	**(± 0.190)**	**2.08**	**(the best in 20.2 %)**
CBS3M_FIFO_ROFF	1.000	1.332	(± 0.227)	2.57	(the best in 0.1 %)
CBS3M_EDF_ROFF	1.000	1.272	(± 0.214)	2.49	(the best in 3.8 %)

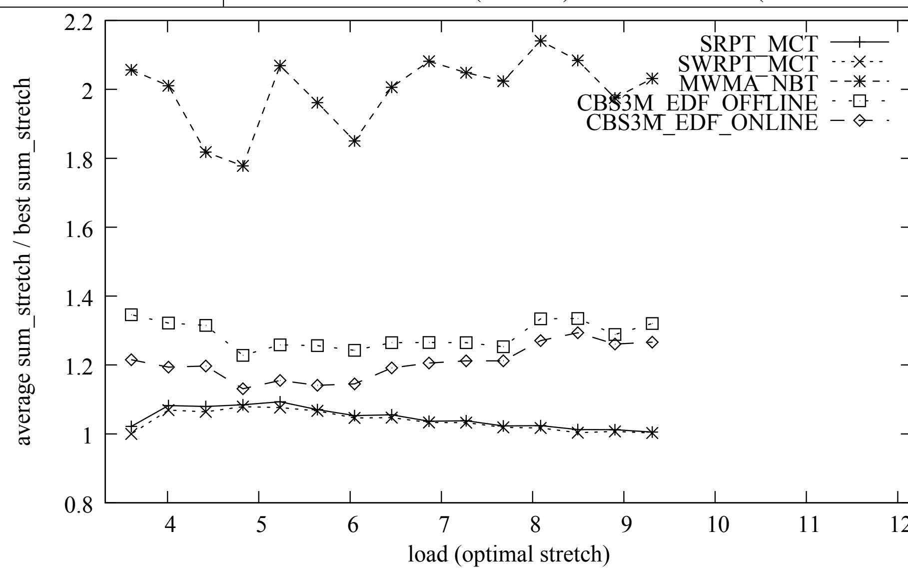


Fig. 4. Sum-stretch of all heuristics in the simulations, and its evolution for the best heuristics in under different load conditions.

References

[1] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Trans. Computers*, 50(10):1052–1070, 2001.

[2] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien. Static tiling for heterogeneous computing platforms. *Parallel Computing*, 25:547–568, 1999.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	1.343	2.716	(± 0.684)	5.31	(the best in 0.0 %)
FIFO_MCT	1.000	1.329	(± 0.202)	2.11	(the best in 0.1 %)
FIFO_DD	1.343	2.716	(± 0.684)	5.31	(the best in 0.0 %)
SPT_RR	1.325	2.714	(± 0.685)	5.33	(the best in 0.0 %)
SPT_MCT	1.000	1.329	(± 0.202)	2.1	(the best in 0.0 %)
SPT_DD	1.325	2.714	(± 0.685)	5.33	(the best in 0.0 %)
SRPT_RR	1.325	2.714	(± 0.686)	5.32	(the best in 0.0 %)
SRPT_MCT	1.000	1.328	(± 0.202)	2.1	(the best in 0.0 %)
SRPT_DD	1.325	2.714	(± 0.686)	5.32	(the best in 0.0 %)
SWRPT_RR	1.322	2.715	(± 0.686)	5.32	(the best in 0.0 %)
SWRPT_MCT	1.000	1.328	(± 0.202)	2.1	(the best in 0.0 %)
SWRPT_DD	1.322	2.715	(± 0.686)	5.32	(the best in 0.0 %)
MWMA_NBT	1.000	1.079	(± 0.070)	1.45	(the best in 4.6 %)
MWMA_MS	1.000	1.078	(± 0.067)	1.42	(the best in 2.1 %)
CBS3M_FIFO_ONLINE	1.000	1.029	(± 0.029)	1.17	(the best in 7.5 %)
CBS3M_EDF_ONLINE	**1.000**	**1.004**	**(± 0.006)**	**1.05**	**(the best in 35.0 %)**
CBS3M_FIFO_ROFF	**1.000**	**1.018**	**(± 0.023)**	**1.22**	**(the best in 17.6 %)**
CBS3M_EDF_ROFF	**1.000**	**1.003**	**(± 0.006)**	**1.07**	**(the best in 53.0 %)**

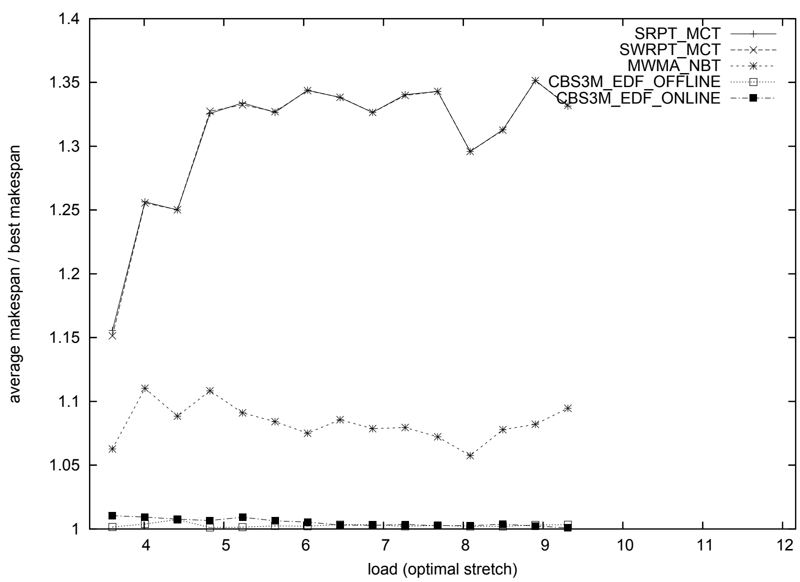


Fig. 5. Makespan of all heuristics in the simulations, and its evolution for the best heuristics in under different load conditions.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	1.146	3.097	(± 1.135)	10.2	(the best in 0.0 %)
FIFO_MCT	**1.000**	**1.281**	**(± 0.258)**	**2.83**	**(the best in 14.4 %)**
FIFO_DD	1.146	3.097	(± 1.135)	10.2	(the best in 0.0 %)
SPT_RR	1.386	3.282	(± 1.222)	10.9	(the best in 0.0 %)
SPT_MCT	1.002	1.460	(± 0.287)	3.09	(the best in 0.0 %)
SPT_DD	1.386	3.282	(± 1.222)	10.9	(the best in 0.0 %)
SRPT_RR	1.386	3.289	(± 1.225)	10.9	(the best in 0.0 %)
SRPT_MCT	1.003	1.473	(± 0.306)	4.28	(the best in 0.0 %)
SRPT_DD	1.386	3.289	(± 1.225)	10.9	(the best in 0.0 %)
SWRPT_RR	1.382	3.291	(± 1.225)	10.9	(the best in 0.0 %)
SWRPT_MCT	1.000	1.477	(± 0.309)	4.28	(the best in 0.1 %)
SWRPT_DD	1.382	3.291	(± 1.225)	10.9	(the best in 0.0 %)
MWMA_NBT	1.000	1.181	(± 0.153)	1.99	(the best in 7.0 %)
MWMA_MS	1.000	1.261	(± 0.189)	2.32	(the best in 1.1 %)
CBS3M_FIFO_ONLINE	1.000	1.054	(± 0.061)	1.52	(the best in 5.8 %)
CBS3M_EDF_ONLINE	**1.000**	**1.031**	**(± 0.057)**	**1.48**	**(the best in 23.2 %)**
CBS3M_FIFO_ROFF	**1.000**	**1.037**	**(± 0.058)**	**1.48**	**(the best in 21.6 %)**
CBS3M_EDF_ROFF	**1.000**	**1.023**	**(± 0.055)**	**1.48**	**(the best in 48.7 %)**

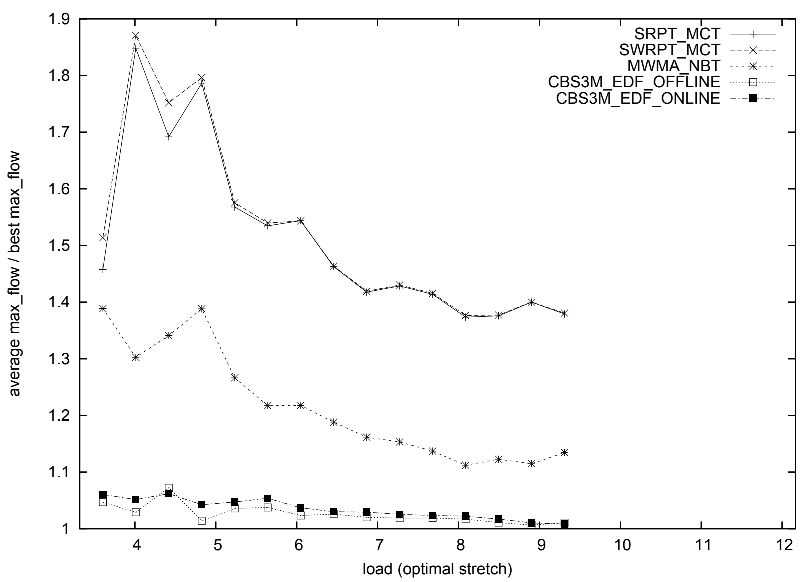


Fig. 6. Max-flow of all heuristics in the simulations, and its evolution for the best heuristics in under different load conditions.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	1.644	4.020	(± 1.567)	16.3	(the best in 0.0 %)
FIFO_MCT	1.134	1.652	(± 0.264)	3.33	(the best in 0.0 %)
FIFO_DD	1.644	4.020	(± 1.567)	16.3	(the best in 0.0 %)
SPT_RR	1.196	2.811	(± 1.081)	9.21	(the best in 0.0 %)
SPT_MCT	1.000	1.149	(± 0.171)	2.32	(the best in 3.5 %)
SPT_DD	1.196	2.811	(± 1.081)	9.21	(the best in 0.0 %)
SRPT_RR	1.079	2.704	(± 1.048)	9.03	(the best in 0.0 %)
SRPT_MCT	**1.000**	**1.105**	**(± 0.151)**	**2.23**	**(the best in 32.1 %)**
SRPT_DD	1.079	2.704	(± 1.048)	9.03	(the best in 0.0 %)
SWRPT_RR	1.079	2.706	(± 1.049)	9.03	(the best in 0.0 %)
SWRPT_MCT	**1.000**	**1.108**	**(± 0.152)**	**2.23**	**(the best in 15.4 %)**
SWRPT_DD	1.079	2.706	(± 1.049)	9.03	(the best in 0.0 %)
MWMA_NBT	1.000	1.404	(± 0.217)	2.29	(the best in 0.1 %)
MWMA_MS	1.359	2.333	(± 0.355)	3.7	(the best in 0.0 %)
CBS3M_FIFO_ONLINE	1.000	1.122	(± 0.101)	1.62	(the best in 1.4 %)
CBS3M_EDF_ONLINE	**1.000**	**1.065**	**(± 0.090)**	**1.53**	**(the best in 35.6 %)**
CBS3M_FIFO_ROFF	1.000	1.120	(± 0.103)	1.67	(the best in 0.3 %)
CBS3M_EDF_ROFF	**1.000**	**1.087**	**(± 0.101)**	**1.66**	**(the best in 18.7 %)**

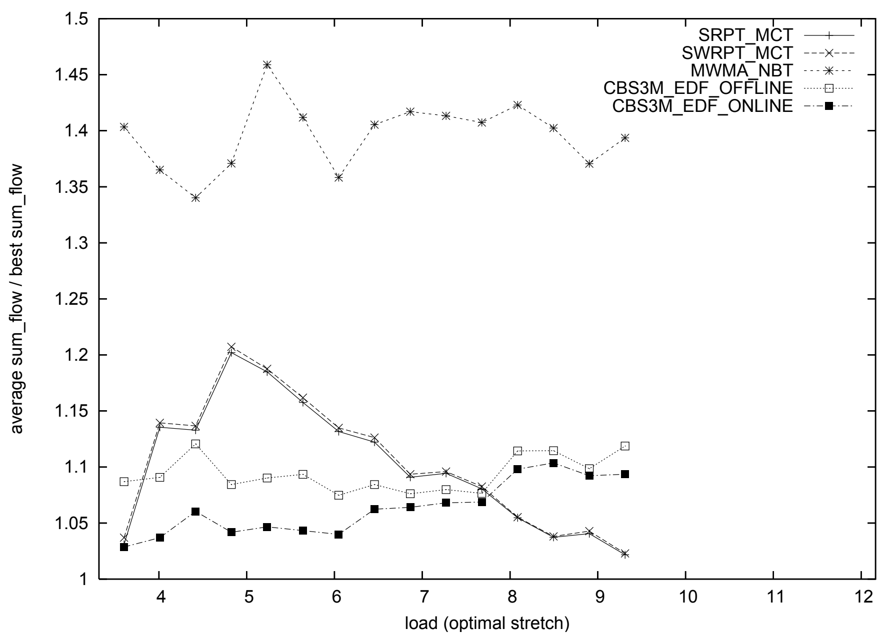


Fig. 7. Sum-flow of all heuristics in the simulations, and its evolution for the best heuristics in under different load conditions.